PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46

A1

(11) International Publication Number: WO 98/44414

(43) International Publication Date: 8 October 1998 (08.10.98)

(21) International Application Number: PCT/US98/05665

(22) International Filing Date: 24 March 1998 (24.03.98)

(30) Priority Data:
08/829,861 31 March 1997 (31.03.97) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: PELEGRI-LLOPART, Eduardo; 1731 Fordham Way, Mountain View, CA 94040 (US). HAMILTON, Graham; 3143 David Court, Palo Alto, CA 94303 (US). KESSLER, Peter, B.; 4121 Verdosa Drive, Palo Alto, CA 94306 (US). WALDO, James, H.; 155 Ruby Road, Dracut, MA 01826 (US). RIGGS, Roger; 4 Briarwood Lane, Burlington, MA 01803 (US). WOLLRATH, Ann, M.; 9 Northwoods Road, Groton, MA 01450 (US).

(74) Agents: HYMAN, Eric, S. et al.; Blakely, Sokoloff, Taylor & Zafman, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025-1026 (US).

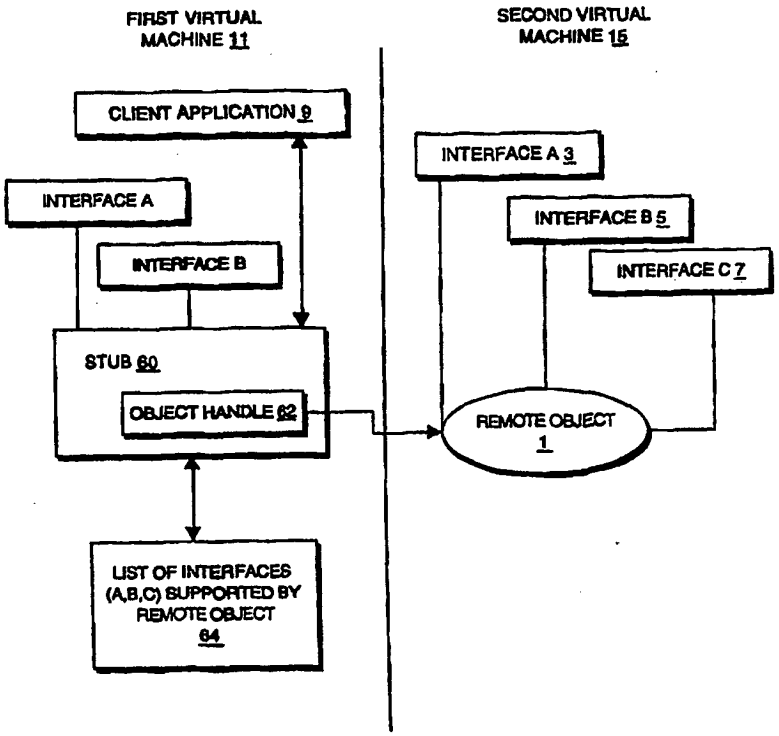
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR GENERATING AND EMPLOYING A RUN-TIME GENERATED STUB TO REFERENCE AN OBJECT IN OBJECT ORIENTED SYSTEMS

(57) Abstract

The method and apparatus for generating and employing a run-time generated stub to reference an object in an object oriented system. This method and apparatus represents in a first virtual machine a remote object implemented in a second virtual machine by employing a stub class. An object reference is sent by the second virtual machine and received by the first virtual machine. The object reference includes an interface descriptor that identifies the interface(s) of the remote object and an object handle that identifies the remote object. At run-time, the information associated with the remote object is transformed into a stub class that represents the remote object and implements only those interfaces identified by the interface descriptor and also defined by the first virtual machine. After the stub class is created, an instance of that stub class is generated and provided to the first virtual machine. In an alternative embodiment, an interface specific stub is created for each interface that is identified by the interface descriptor and defined by the first virtual machine. The interface specific stub is created at compile time. A delegator stub class is created at run-time that delegates to the interface specific stub(s).



METHOD AND APPARATUS FOR GENERATING AND EMPLOYING A RUN-TIME GENERATED STUB TO REFERENCE AN OBJECT IN OBJECT ORIENTED SYSTEMS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to the fields of distributed computer systems, client-server computing, and object oriented programming. Specifically, the present invention relates to a **method** and apparatus for clients on different machines to communicate references to remote objects such that the receiving machine has as much information as wanted, neither more nor less, of the type of the remote object.

Background

In client-server computing, typically there is a set of computers that can communicate with one another through a network connecting the computers. A first computer in the network that is referred to as a client sends a request to a second computer in the network that is referred to as a server. In response to the request, the server takes action and generally provides data representing the results of the action to the client. The client-server model also generalizes to the case where distinct processes, running on the same computer, are communicating with one another through some protected mechanism and are acting as server and clients.

For examples of client-server systems see: (i) J.K. Ousterhout et al, "Medusa: An Experiment in Distributed Operating System Structure", Communications of the ACM 23(2), 1980; (ii) R.M. Needham and A.J. Herbert, "The Cambridge Distributed Computing System", Addison-Wesley, 1982; and (iii) J.K. Ousterhout et al "The Sprite Distributed Operating System", IEEE Computer, 1988.

a class hierarchy. Each class can have one or more subclasses, each of which can add attributes and/or member functions and can redefine member functions. For example, a desk class is a subclass of the table class and defines an additional attribute which specifies the number of drawers of the desk. Thus, in addition to an attribute which defines the number of drawers of the desk, a desk inherits from the table class the color attribute and the attributes specifying various dimensions of the desk. Furthermore, a desk can perform the weight bearing member function described above and the behavior of the desk in performing the weight bearing member function is defined by the table class. The desk class can define additional member functions not defined by the table class, e.g., an open drawer member function, a place object in drawer member function, a remove object from drawer member function, and a close drawer member function. As described above, a subclass can also redefine an inherited member function, thereby superseding the inherited definition of the member function. For example, the table class can have a subclass which is a folding table class. The weight bearing member function described above can be redefined by the folding table class such that a folding table asked to bear a weight is verified to be in an unfolded, upright position prior to bearing any weight.

A member function defined by a class is performed by directing an object of the class to perform the member function. The particular member function performed is selected according to the class to which the object belongs. If the class to which the object belongs does not define such a member function, the member function is selected from the superclass which defines the member function and for which no subclass defines the member function. A first class is a superclass of a second class if the second class is a subclass of the first class. The member function is performed in the context of the state of the object directed to perform the member function.

Another helpful feature of object-oriented environments is referred to as interfaces. An interface is a collection of member function definitions without implementation. A programmer can employ interfaces to specify the member functions an object can perform. An interface defines the inputs, outputs, attributes, and exceptions of the member functions without the implementation. Implementations of a member function are the computer instructions that are executed by the computer to perform the member functions. While an interface specifies the inputs, outputs, attributes and exceptions of a member function, the interface does not include any computer instructions (i.e., the implementation of the member function). An object can have one or more interfaces. A class that specifies an interface typically includes implementation to perform all the member functions of the interface.

When the remote object includes multiple interfaces, a problem arises as to what extent the stub object on the local machine describes the interfaces of the remote object. Prior art systems address this problem using one of the following approaches.

In a first approach, the stub class in the local machine defines only the interface of the remote object that is explicitly requested prior to invoking a member function of that particular interface on the local machine. At any given time, the stub class (hereinafter referred to as the interface-specific stub class) only implements one of the interfaces of the remote object denoted by the stub. One disadvantage of this first approach is that any object instantiated from this stub class inaccurately responds to queries as to interface type on interface different to the one specifically requested.

A software language construct can be employed to query an object as to the object's interfaces (also referred to as interface type). For example, in JAVA the command instanceof (obj) yields a true or false to indicate

having to export unnecessary implementation of interfaces not used by the local machine.

SUMMARY OF THE INVENTION

In accordance with the teachings of the present invention, a stub class is employed to represent in a first virtual machine a remote object implemented in a second virtual machine. The first virtual machine receives a reference to the remote object. This reference includes information associated with the remote object and also includes a list of interfaces of the remote object. The information associated with the remote object can include information to uniquely identify a location where the implementation of the remote object resides. After receiving the reference to the remote object, the stub class generator, in accordance with the present invention, transforms at run-time the information associated with the remote object into a stub class that represents the remote object in the first virtual machine and implements in the first virtual machine only those interfaces on the list of interfaces that are also supported by the first virtual machine.

The stub class is generated from the information associated with the remote object by comparing the list of interfaces, included in the reference, with those interfaces defined in the first virtual machine. The stub class generator generates a common interface list that includes those interfaces that are on the list of interfaces and are also supported by the first virtual machine. The stub class generator also generates a list of member functions (herein referred to as a member function list) supported by the common interface list. The stub class is generated from the common interface list, the member function list, and the information associated with the remote object received from the second virtual machine. Objects of this stub class are employed by a process to invoke member functions of the remote object.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects, features and advantages of the system of the present invention will be apparent from the following description in which:

Figure 1 illustrates one embodiment of the present invention which employs a run-time stub to represent in a first virtual machine an object implemented in a second virtual machine.

Figure 2 illustrates an alternative embodiment of the present invention which employs a delegator stub to represent in a first virtual machine an object implemented in a second virtual machine.

Figure 3 illustrates a block diagram of a distributed system including a client machine and a server machine, according to one embodiment of the present invention.

Figure 4 is a flowchart illustrating the operation of the client machine shown in **Figure 3**.

Figure 5 is a flowchart illustrating the operation of the server machine shown in **Figure 3**.

Figure 6 is a flowchart illustrating the method of unmarshaling an argument or result of the present invention.

Figure 7 is a flowchart illustrating the method of marshaling an argument or result of the present invention.

Figures 8 and **9** illustrates in greater detail the run-time stub generator of the present invention.

Figure 10 is a flowchart illustrating a general overview of the operation of the run-time stub generator.

Figure 11 illustrates a block diagram of a distributed system including a client machine and a server machine, according to an alternative embodiment of the present invention.

An embodiment of the present invention employs an object reference having an interface descriptor (e.g., a list) of the interfaces supported by the implementation of the remote object. This interface descriptor is sent together with other object reference information including an object handle to the first virtual machine. The object reference is passed from one virtual machine to another. The present invention describes interfaces in the following ways.

Describing an Interface

One way to describe an interface is by just giving its name. This requires or assumes that the name of the interface is enough to uniquely identify it.

An alternative mechanism is the technique described in the patent application titled, "System and Method for Generating Identifiers for Uniquely Identifying Object Types Used in Processing of Object-Oriented Programs and the Like", invented by James H. Waldo, Krishna Bharat and Roger Riggs, and assigned to the assignee of the present application. In this mechanism, the name of the class and a fingerprint of it are used. The fingerprint assures (with confidence depending on the size of the fingerprint) that the sending and receiving machines have the same definitions for the interface they are referring.

Overview of the Present Invention

An embodiment of the present invention employs the list of interfaces to generate an object whose type is as close as possible to that of the original remote object, but the object does not require any additional types external to the receiving machine. The generation of this object relies on the ability to construct, at run-time a new stub class embodying the object

interface-specific stubs includes a copy of the object handle 71 associated with remote object 1.

General Run-Time Stub Generation Mechanism

Figure 3 illustrates a block diagram of a distributed system including a first virtual machine 11 (e.g., a local machine) and a second virtual machine 15 (e.g., a remote machine), according to one embodiment of the present invention. The operation of this embodiment of the present invention will be illustrated with reference to the flowcharts of **Figure 4** and **Figure 5**.

In **Figure 3**, the client application 400 calls the client stub 402. The client stub performs the steps shown in **Figure 4** and then returns to the client application. The client stub is implemented on top of the client marshal buffer routines 408, which interact with the network 422, with the object descriptor generator 442 and with the run-time stub generator and the run-time stub generator 410.

Figure 4 is a flowchart illustrating the operation of a client stub 402 in **Figure 3**. The stub first marshals all arguments 460, then it marshals and invokes the member function 462. At this moment, the remote member function is actually performed. On its completion, the client stub proceeds to unmarshal the result of the member function, or its exception 464, and then propagates this result or exception to the client application 466.

The client marshal buffer routines 408 can call a run-time stub generator 410 and object descriptor generator 442 according to the present invention. Equally, the server marshal buffer routines 463 can call the run-time stub generator 410 and object descriptor generator 442. Both object descriptor generator 442 and run-time stub generator 410 are described in greater detail hereinafter.

as well as, an identification specifying where on that machine the object resides.

In step 496, a run-time stub generator 410 of the present invention obtains or generates a stub object from the interface descriptor (e.g., list of interfaces). In step 498, the present invention assigns the object handle to the stub object. In step 500, the stub object is returned to the client application 400. The run-time stub generator 410 of the present invention is described in greater detail hereinafter with reference to **Figures 8, 9 and 10.**

Figure 7 is a flowchart illustrating the method of marshaling an argument or result of the present invention. This method of marshaling an argument or result of the present invention may be implemented in step 460 of **Figure 4** and step 478 of **Figure 5.**

In step 510, a marshal routine of the present invention determines if an argument or result is a reference to a remote object. If no, the processing proceeds to a typical marshal routine 511. For an example of prior art marshaling and unmarshaling techniques, please see *"Implementing Remote Procedure Calls"*, by Andrew D. Birrel, Bruce Jay Nelson, in ACM Transactions on Computer Systems (TOCS), Vol. 2, Number 1, February 1984, pp. 39-59. If yes, the marshal routine of the present invention, in step 512, determines if a remote object is local to that virtual machine. If yes, in step 514 the marshal routine of the present invention obtains a list of remote interfaces from the actual implementation of the object on the first virtual machine. In step 516, the marshal routine of the present invention creates an object handle for the actual implementation of the object. In steps 518 and 519, the marshal routine of the present invention marshals the list of interfaces and the object handle.

If the query of decision block 512 is no, in step 520 the marshal routine of the present invention extracts a list of remote interfaces from a field in

common interface list and the member function list at run-time. In other words, the member function implementation unit 634 of the stub class generator 620 of the present invention generates at run-time a stub class that represents in the first virtual machine (the client) the remote object which is implemented in a second virtual machine (e.g., server) based on the common interface list and the member function list.

A stub class instantiation mechanism 636 receives the stub class, the interface descriptor and the object handle and generates an instance of the stub class. In step 818, the present invention instantiates the stub class and creates an instance of the stub class based upon the stub class, the interface descriptor, and the object handle.

Referring again to **Figure 8**, the run-time stub class generator 620 includes a member function list generator 630 and a list implementation unit 634.

The member function list generator 630 includes an input for the common interface list 612. Based on the common interface list 612, the member function list generator 630 generates a member function list corresponding to all the common interfaces. A member function implementation unit 634 is coupled to the member function list generator 630. The member function implementation unit 634 receives the member function list from the member function list generator 630. The implementation of a member function for a stub instance uses the information on its object handle and involves marshaling the arguments in a form that can be used to send across machines, invoking the remote member function, and then unmarshaling results (possibly including exceptions). The implementation of a stub member function can be determined, using known techniques, from the signature of the member function. The signature of a member function includes the name of the

generate the stub class corresponding to the common interface list and then the cache is updated.

Figure 9 illustrates the instance creation mechanism employed by the present invention. After the stub class is generated, the instance creation mechanism (shown in **Figure 9**) is invoked. The instance creation mechanism, which is well known in the art, includes an input for receiving the stub class, and in response, generates an instance (i.e., an object) of that stub class.

Figure 10 is a flowchart illustrating a general overview of the operation of the run-time stub generator. In step 803, the run-time stub generator receives an interface descriptor (e.g., a list of interfaces) of the remote object. In processing step 803, the run-time stub generator receives an object handle associated with the remote object. In processing step 808, the run-time stub generator compares the interfaces of the remote object identified by the interface descriptor with the interfaces supported by the first virtual machine (i.e., the local machine). In processing step 810, the run-time stub generator generates a common interface list that includes interfaces identified by the interface descriptor and also supported by the first virtual machine. In processing step 812, the run-time stub generator generates a member function list that includes the member functions defined by the interfaces listed in the common interface list. In processing step 816, the run-time stub generator generates a stub class that represents in the local machine the remote object that is implemented in the second virtual machine based on the common interface list and the member function list. In processing step 818, an instance of the stub classes having the interface descriptor and the object handle is created.

invention may be applied to a plurality of interface-specific stubs with each interface-specific stub corresponding to a particular interface. In step 1002, the interface-specific stub 908, 912 marshals all the arguments. The interface-specific stub 908, 912 calls the client marshal buffer routines 918. In step 1004, the client marshal buffer routines 918 marshal and invoke the member function using a marshal buffer. The marshal buffer is passed to the server machine 940 via the network 922. The client marshal buffer routines 918 may call the run-time delegator stub generator 910 of the present invention, which will be described in greater detail hereinafter.

In step 1008, the client marshal buffer routines 918 unmarshal the result or exception In step 1010, the interface-specific stub returns to the delegator stub 902. In step 1012, the delegator stub 902 returns the result to the client application 900 or raises an exception.

The operation of the server 940 of **Figure 11** is illustrated in **Figure 5**. Moreover, the unmarshal routine of the present invention, as illustrated in **Figure 6**, may be implemented in step 1008 of **Figure 12**. Also, the marshal routine of the present invention may be implemented in step 1002 of **Figure 12**.

Figure 13 illustrates a block diagram of a run-time delegator stub generator and a pre-run-time interface specific stub generator of the present invention. A run-time delegator stub generator 910 has an input for receiving an interface descriptor 602 from the server machine 940. The run-time delegator stub generator 910 generates a run-time generator delegator stub 902 based upon the interface descriptor 602 and the object handle. The run-time delegator stub generator 910 will be described in greater detail hereinafter with respect to **Figures 14-16**.

The pre-run-time interface specific stub generator 1020 includes an input for receiving a specific interface 1022. The pre-run time interface

In step 1418, the present invention generates at run-time a delegator stub class that delegates to interface-specific stubs that are generated at compile time. The delegator stub and the interface-specific stub together representing in the first virtual machine the remote object which is implemented in a second virtual machine based on the common interface list and the member function list.

In step 1420, the present invention instantiates the delegator stub class with the interface descriptor and the object handle.

Figure 16 describes in greater detail the additional steps of processing step 1418 of **Figure 15**.

In step 1412, the delegator class generation mechanism 1030, that receives a list of common interfaces from the common interface list generator 610, generates a list of member functions that are supported by the common interfaces. Specifically, the member function list generator 1050 of the delegator class generation mechanism 1030 generates this list of member functions. In step 1414, the member function-interface assignment unit 1054 assigns each member function of the member function list to one of the common interfaces. In step 1491, the field generator unit, 1054, generates one field per interface in the list of common interfaces. The fields are of type reference to the interface-specific stubs, specifically that stub corresponding to the interface being considered.

In step 1492, the member function generator unit 1056 generates the implementation for each member function in the member function list. The member function generator 1056 of the present invention generates a member function for the delegator stub having the same signature (e.g., the same name, arguments, return type, and exception list) as the member function in the member function list. This new member function invokes the member function of the same name of the object referred to by the field

```
INTERFACE_A_STUB.Afoo (    );

                }
Cfoo(  )       {
INTERFACE_B_STUB.Cfoo (    );
                }
```

Figures 18 and **19** are flowcharts for how the present invention employs the interface descriptor table. **Figure 18** describes the operations performed by block 519. If a machine A wants to send an interface D to machine B, then it first checks if it has already done so (decision block 1804). This is done by consulting the Interface Descriptor Table. If the interface descriptor D has never been sent between A and B, then a new tag T is created (processing step 1810). In processing step 1812, the Interface Descriptor Table is updated to record the association of T with the transmission of D from A to B. In processing steps 1814, 1816, the tag T and the interface description D are marshaled and sent across the network. If, on the other hand, the interface descriptor D has previously been sent between A and B, then the Interface Description Table is consulted, the tag T previously used is extracted (processing step 1806), and only T is marshaled and sent across the network (processing step 1808).

Figure 19 describes the operations performed by block 803 in **Figure 10**; they correspond to the Interface Descriptor Table Check Unit 606 and the table 608. **Figure 19** is a flowchart illustrating the step of machine B receiving an interface descriptor D from machine A. In processing step 1904, machine B receives tag (T) from the network. In decision block 1908, a

A data storage device 1707 such as a magnetic disk or optical disk and its corresponding disk drive can be coupled to computer system 1700. Computer system 1700 can also be coupled via bus 1701 to a display device 1721, such as a cathode ray tube (CRT), for displaying information to a computer user. An alphanumeric input device 1722, including alphanumeric and other keys, is typically coupled to bus 1701 for communicating information and command selections to processor 1702. Another type of user input device is cursor control 1723, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 1702 and for controlling cursor movement on display 1721. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane.

Alternatively, other input devices such as a stylus or pen can be used to interact with the display. A displayed object on a computer screen can be selected by using a stylus or pen to touch the displayed object. The computer detects the selection by implementing a touch sensitive screen. Similarly, a light pen and a light sensitive screen can be used for selecting a displayed object. Such devices may thus detect selection position and the selection as a single operation instead of the "point and click," as in a system incorporating a mouse or trackball. Stylus and pen based input devices as well as touch and light sensitive screens are well known in the art. Such a system may also lack a keyboard such as 1722 wherein all interface is provided via the stylus as a writing instrument (like a pen) and the written text is interpreted using optical character recognition (OCR) techniques.

The present invention is related to the use of computer system 1700 to represent a remote object implemented in a second virtual machine by employing a routine stub or a delegator stub. According to one

In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the present invention. Thus, the present invention is not limited to any specific combination of hardware circuitry and software, nor to any particular source for the instructions executed by computer system 1700.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will however be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are accordingly to be regarded as illustrative rather than restrictive.

b) generating a common interface list, said common interface list including those interfaces defined by the first virtual machine and identified by said interface descriptor; and

c) generating said stub class based on said common interface list.

3. The method for representing in a first virtual machine a remote object implemented in a second virtual machine of Claim 2, wherein the step of generating said stub class based on said common interface list further comprises the steps of:

a) generating a member function list, said member function list including the member functions defined by the common interfaces;

b) for each member function in said member function list generating a first sequence of code for marshaling the argument into a marshal buffer, a second sequence of code for invoking a corresponding member function on the remote object, a third sequence of code for unmarshaling one of a return value or exception, and a fourth sequence of code for returning one of said return value or said exception; and

c) generating said stub class at run-time, said stub class based on the common list of interfaces and the member function list, said stub class including code to implement the member function of said member function list.

4. The method for representing in a first virtual machine a remote object implemented in a second virtual machine of Claim 1, wherein said interface descriptor includes a list of interfaces.

5. The method for representing in a first virtual machine a remote object implemented in a second virtual machine of Claim 2,

sequences of instructions from a host computer to a target computer, the sequence of instructions including instructions which, when executed on the target computer, cause the target computer to perform the steps of:

a) receiving an object reference to said remote object from said second virtual machine, said object reference having information associated with said remote object, said information including an interface descriptor and an object handle of said remote object, said object handle for identifying said remote object, said interface descriptor identifying an interface of said remote object implemented in the second virtual machine;

b) transforming at run-time said information associated with the remote object into a stub class, said stub class representing said remote object and implementing only those interfaces identified by said interface descriptor and defined by the first virtual machine; and

c) instantiating said stub class and providing the first virtual machine with an instance associated with said stub class, said instance including said interface descriptor and said object handle.

10. The method for enabling a target computer to represent in a first virtual machine a remote object implemented in a second virtual machine by employing a stub class of Claim 9, wherein the step of transforming at run-time the information associated with the remote object into a stub class further comprises the steps of:

a) comparing said list of interfaces with interfaces defined by the first virtual machine;

b) generating a common interface list, said common interface list including those interfaces defined by the first virtual machine and identified by said interface descriptor; and

generating said stub class based on said common interface list further comprising the steps of:

a) generating a member function list, said member function list including the member function defined by the common interfaces;

b) generating at least one interface specific stub class at compile time based on the common interface list, the member function list and the object handle associated with the remote object; and

c) generating a delegator stub class at run-time based on the common list interfaces and the member function list, said delegator stub class delegating to said interface specific stub.

14. The method for enabling a target computer to represent in a first virtual machine a remote object implemented in a second virtual machine by employing a stub class of Claim 13, further comprising the steps of

a) instantiating said interface specific stub class based on the interface descriptor and the object handle and providing the first virtual machine with an instance associated with said stub class; and

b) instantiating said delegator stub class based on the interface descriptor and the object handle and providing the first virtual machine with an instance associated with said delegator stub class.

15. The method for enabling a target computer to represent in a first virtual machine a remote object implemented in a second virtual machine by employing a stub class of Claim 13, wherein said interface specific stub includes code to implement all the member function in said member function list.

causes said processor to transform at run-time said information associated with the remote object into a stub class, further includes:

a) a fourth sequence of instructions which, when executed by said processor, causes said processor to compare said list of interfaces with interfaces defined by the first virtual machine;

b) a fifth sequence of instructions which, when executed by said processor, causes said processor to generate a common interface list, said common interface list including those interfaces defined by the first virtual machine and identified by said interface descriptor; and

c) a sixth sequence of instructions which, when executed by said processor, causes said processor to generate said stub class based on said common interface list.

19. The computer software product of Claim 18, wherein the sixth sequence of instructions which, when executed by said processor, causes said processor to generate said stub class based on said common interface list, further includes:

a) a seventh sequence of instructions which, when executed by said processor, causes said processor to generate a member function list, said member function list including the member functions defined by the common interfaces;

b) an eighth sequence of instructions which, when executed by said processor, causes said processor, for each member function in said member function list, to generate a first sequence of code for marshaling the argument into a marshal buffer, a second sequence of code for invoking a corresponding member function on the remote object, a third sequence of code for unmarshaling one of a return value or exception, and a fourth

22. The computer software product of Claim 21, further comprising:

a) a thirteenth sequence of instructions which, when executed by said processor, causes said processor to instantiate said interface specific stub class based on the interface descriptor and the object handle and providing the first virtual machine with an instance associated with said stub class; and

b) a fourteenth sequence of instructions which, when executed by said processor, causes said processor to instantiate said delegator stub class based on the interface descriptor and the object handle and providing the first virtual machine with an instance associated with said delegator stub class.

23. A computer system comprising:

a) a processor;

b) a memory, operatively coupled to said processor; and

c) a remote object reference mechanism, operatively coupled to said processor and memory, said remote object reference mechanism including:

a first module, configured to receive an object reference to a remote object, said object reference having information associated with said remote object, said information including an interface descriptor and an object handle of said remote object, said object handle identifying said remote object, said interface descriptor identifying an interface of said remote object;

a second module, operatively coupled to said first module, which when executed by said processor, causes said processor to transform at run-time said information associated with the remote object into a stub class, said stub class representing said remote object and implementing only those

b) an eighth module, operatively coupled to said seventh module which, when executed by said processor, causes said processor, for each member function in said member function list, to generate a first sequence of code for marshaling the argument into a marshal buffer, a second sequence of code for invoking a corresponding member function on the remote object, a third sequence of code for unmarshaling one of a return value or exception, and a fourth sequence of code for returning one of said return value or said exception; and

c) a ninth module, operatively coupled to said eighth module which, when executed by said processor, causes said processor to generate said stub class at run-time, said stub class based on the common list of interfaces and the member function list, said stub class including code to implement the member function of said member function list.

26. The computer system of Claim 23, wherein said interface descriptor is a list of interfaces.

27. The computer system of Claim 23, wherein the remote object reference mechanism further includes:

a) a tenth module, operatively coupled to said ninth module which, when executed by said processor, causes said processor to generate a member function list, said member function list including the member function defined by the common interfaces;

b) an eleventh module, operatively coupled to said tenth module which, when executed by said processor, causes said processor to generate at least one interface specific stub class at compile time based on the common interface list, the member function list and the object handle associated with the remote object; and

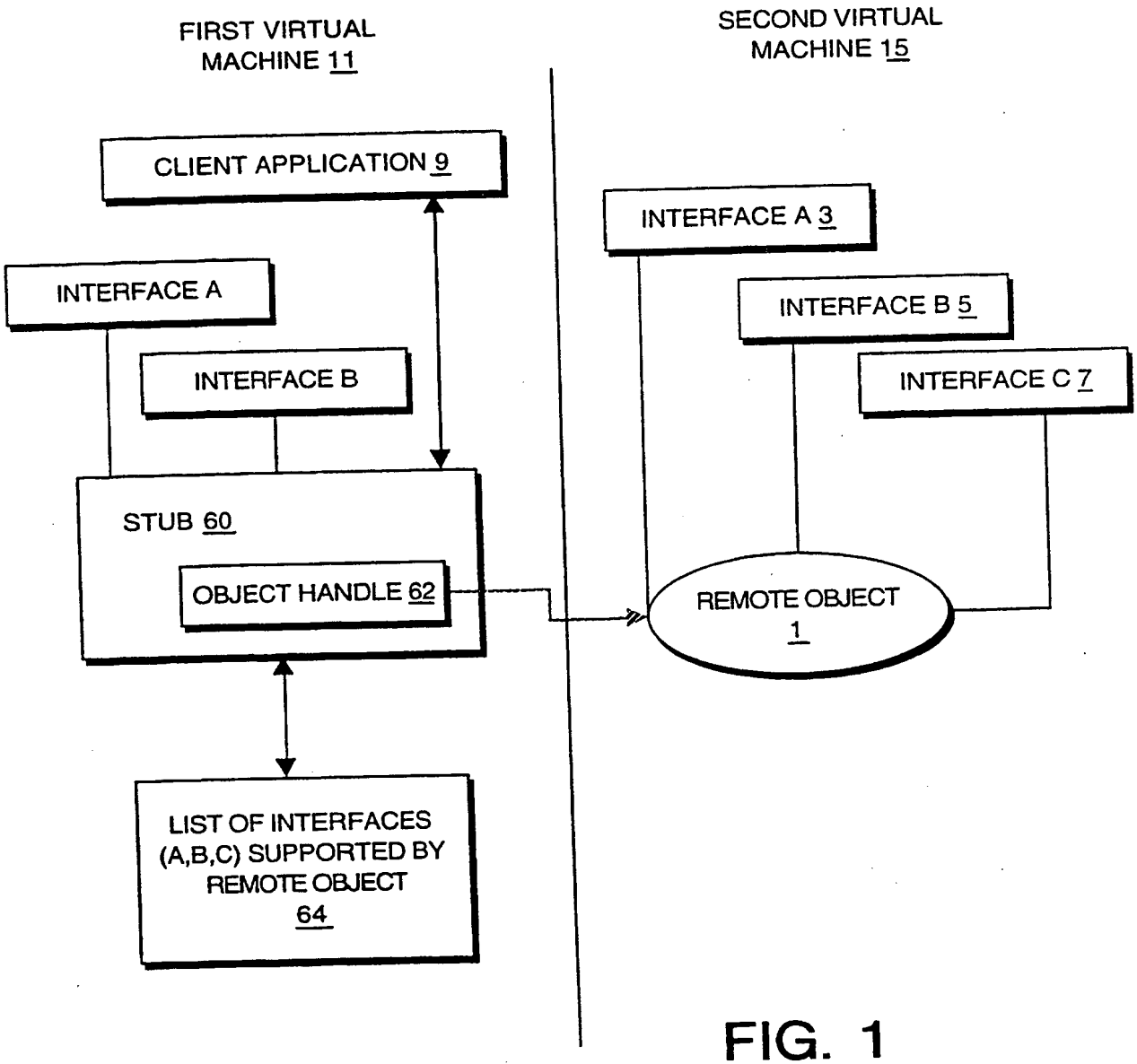


FIG. 1

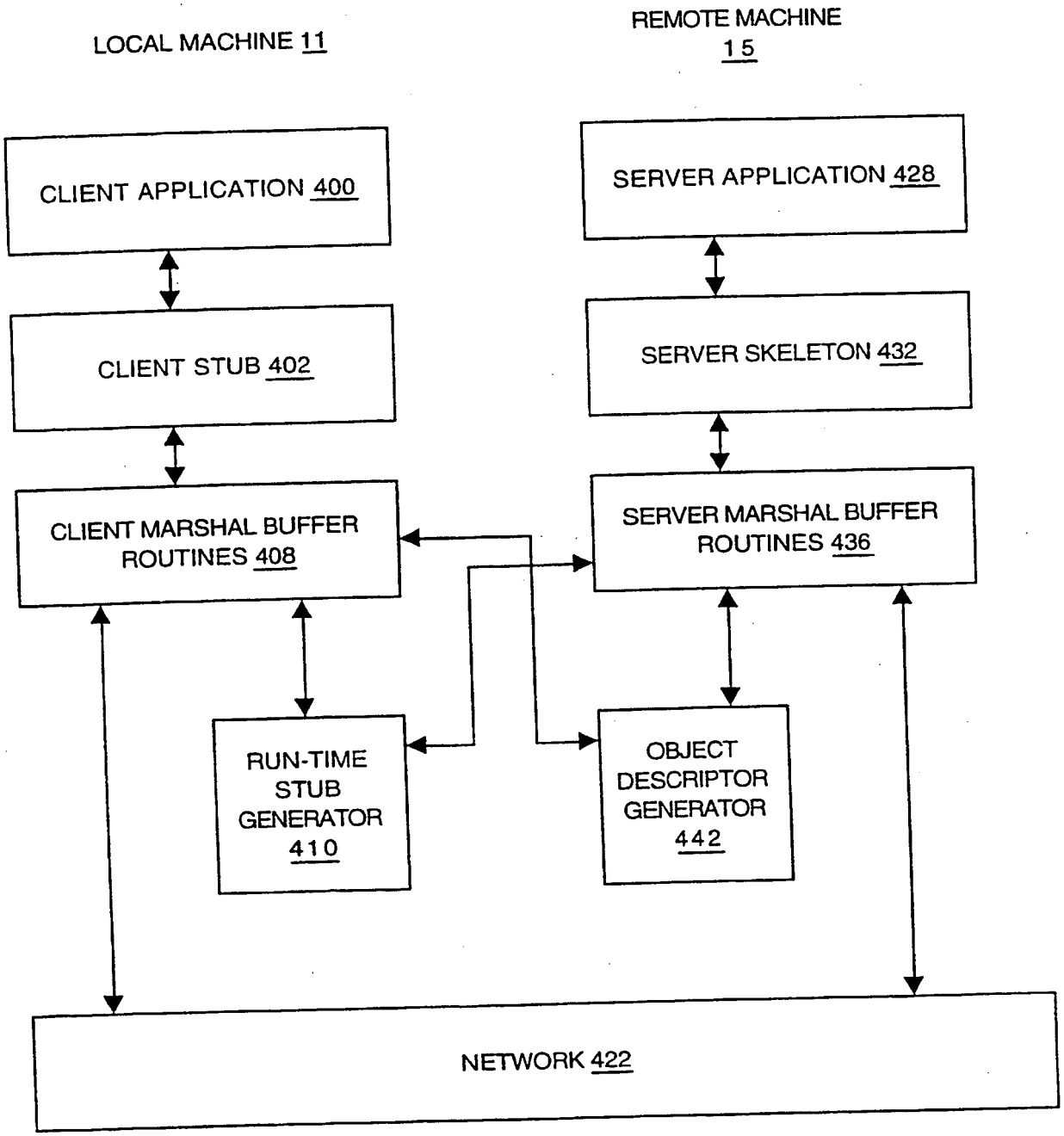


FIG. 3

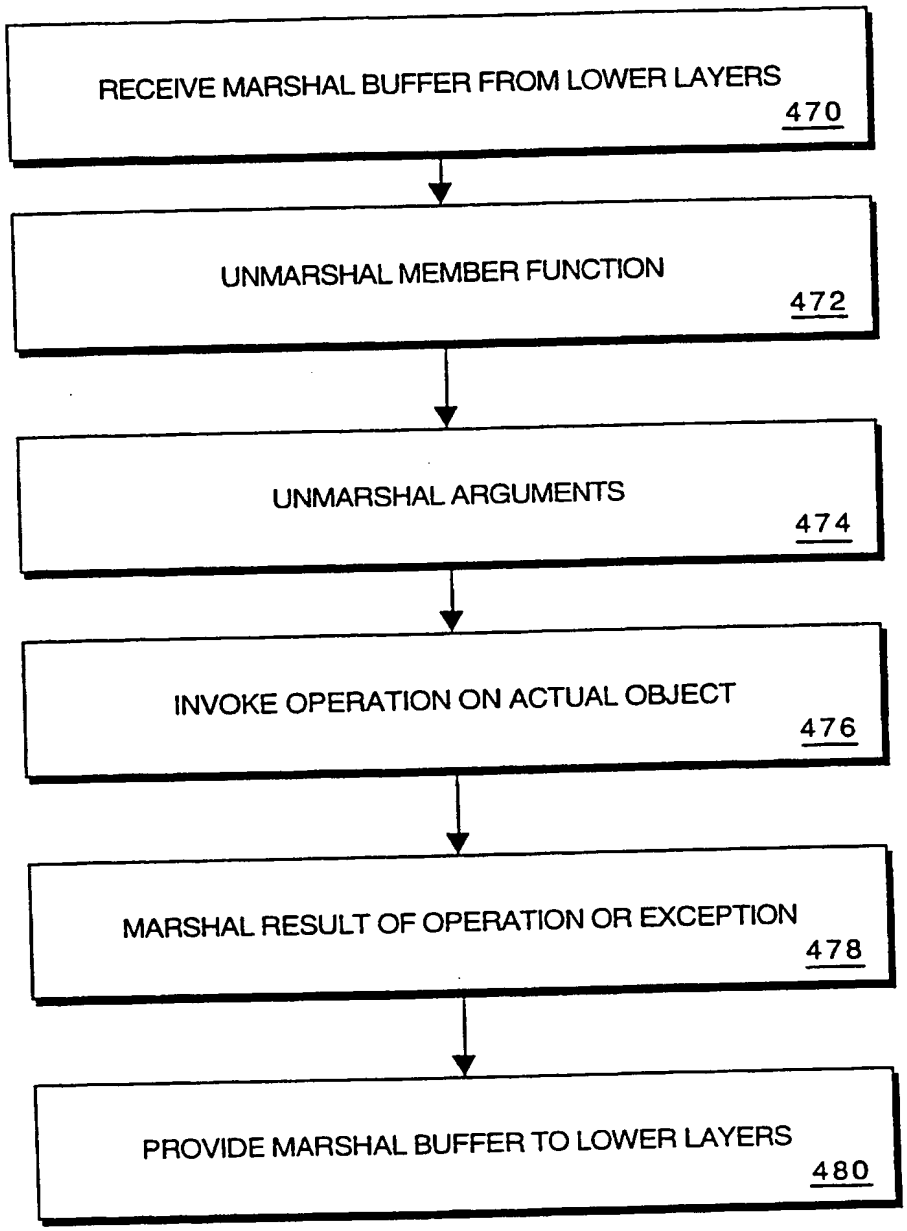


FIG. 5

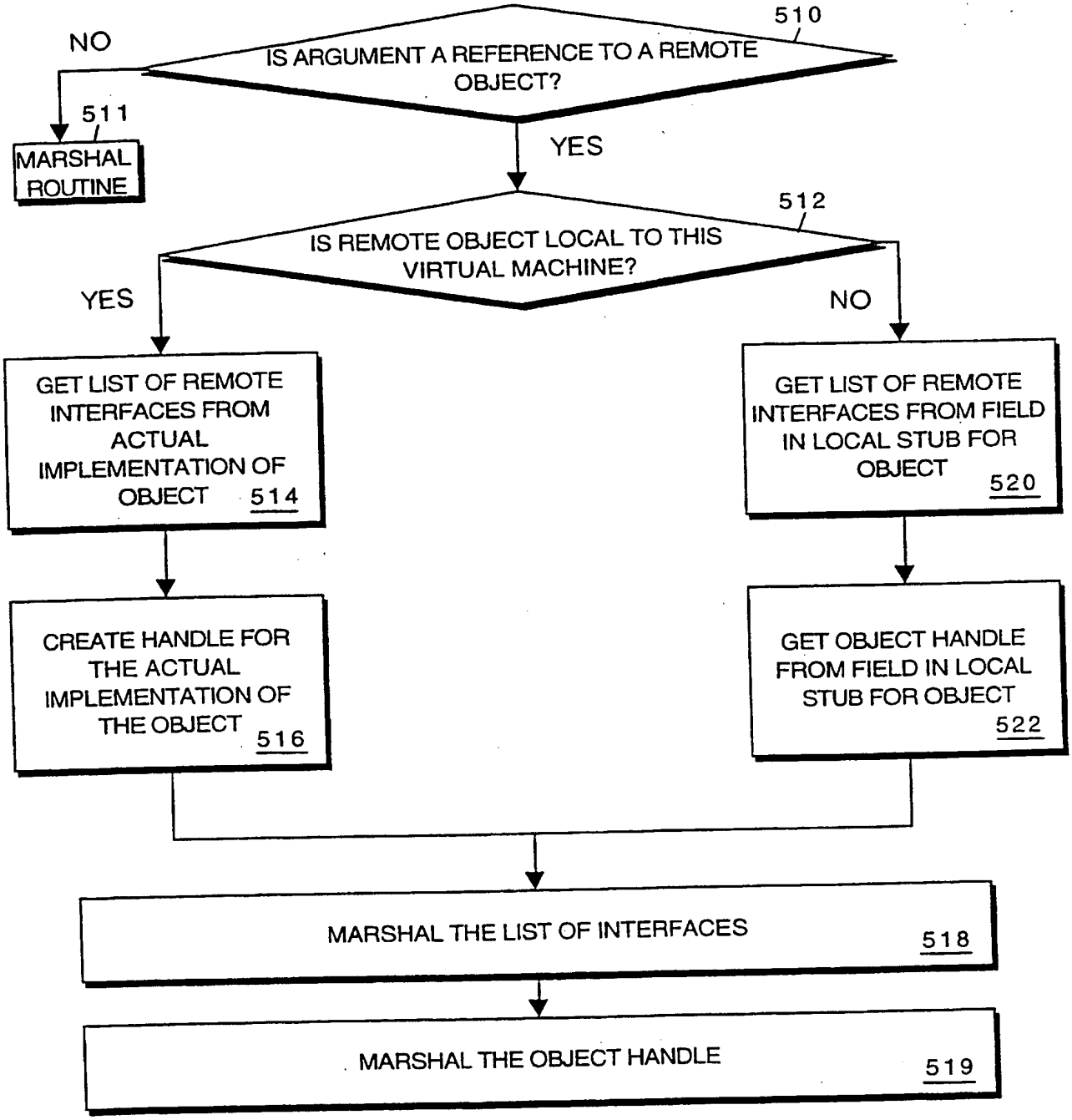


FIG. 7

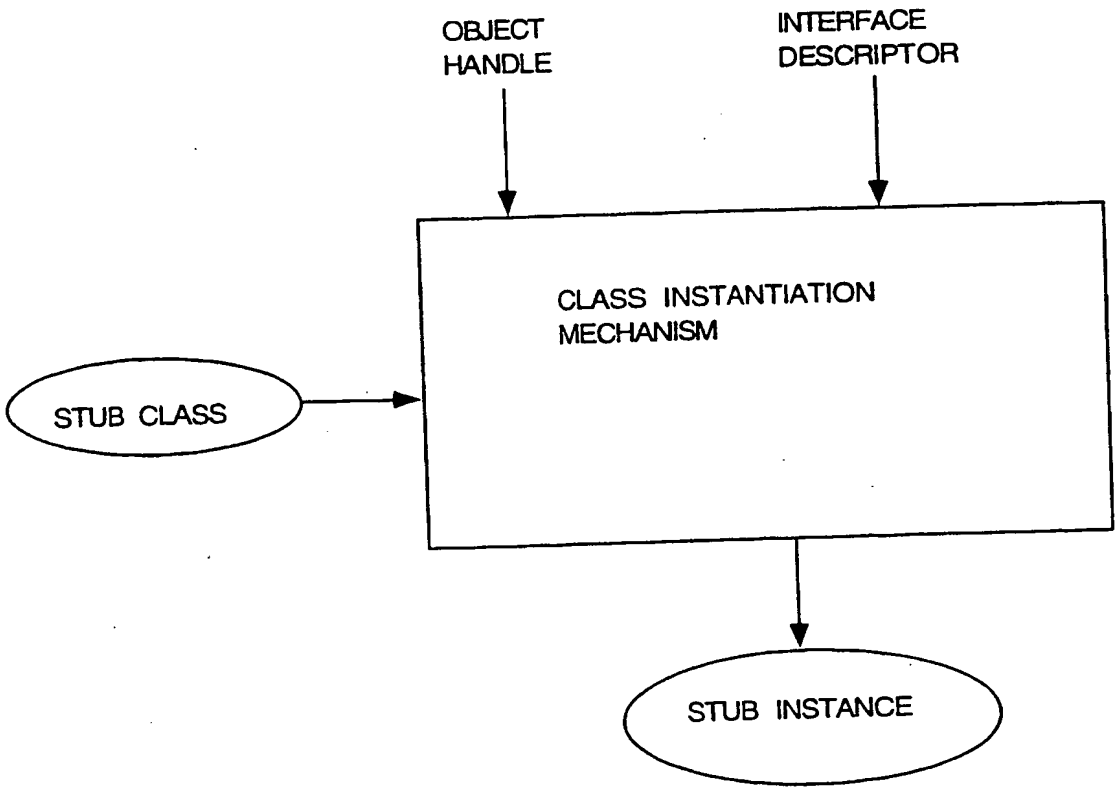


FIG. 9

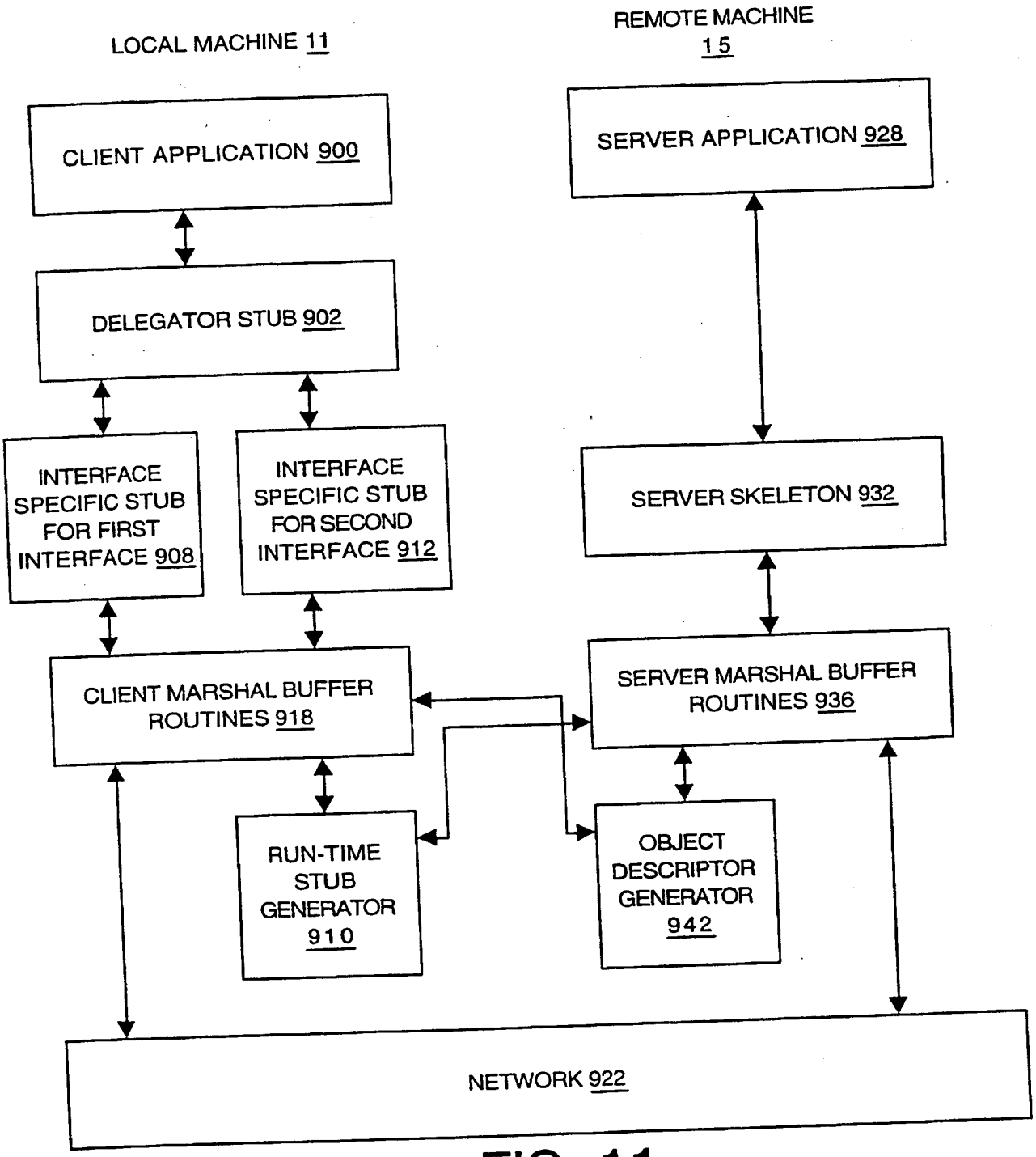


FIG. 11

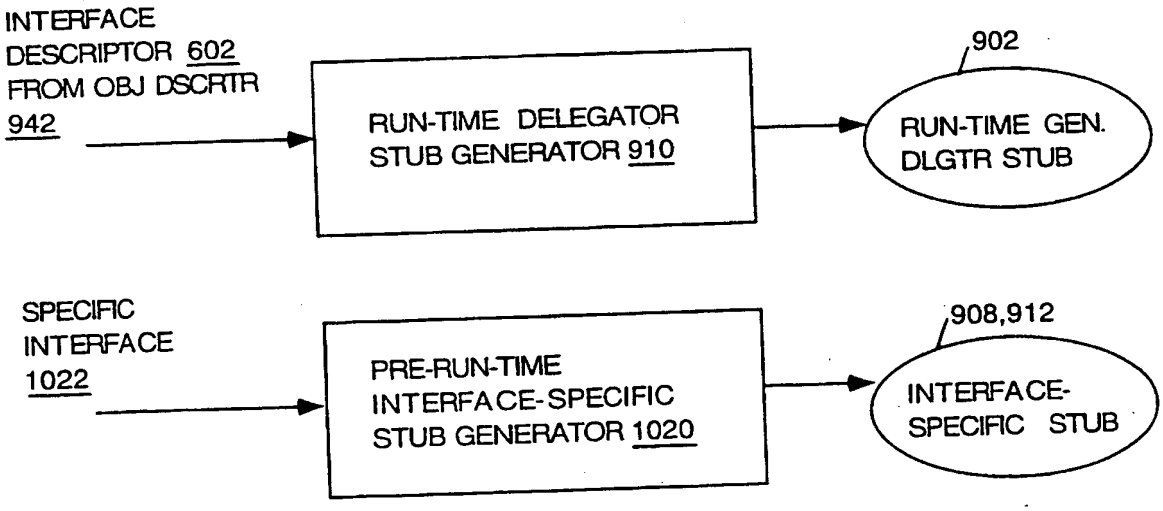


FIG. 13

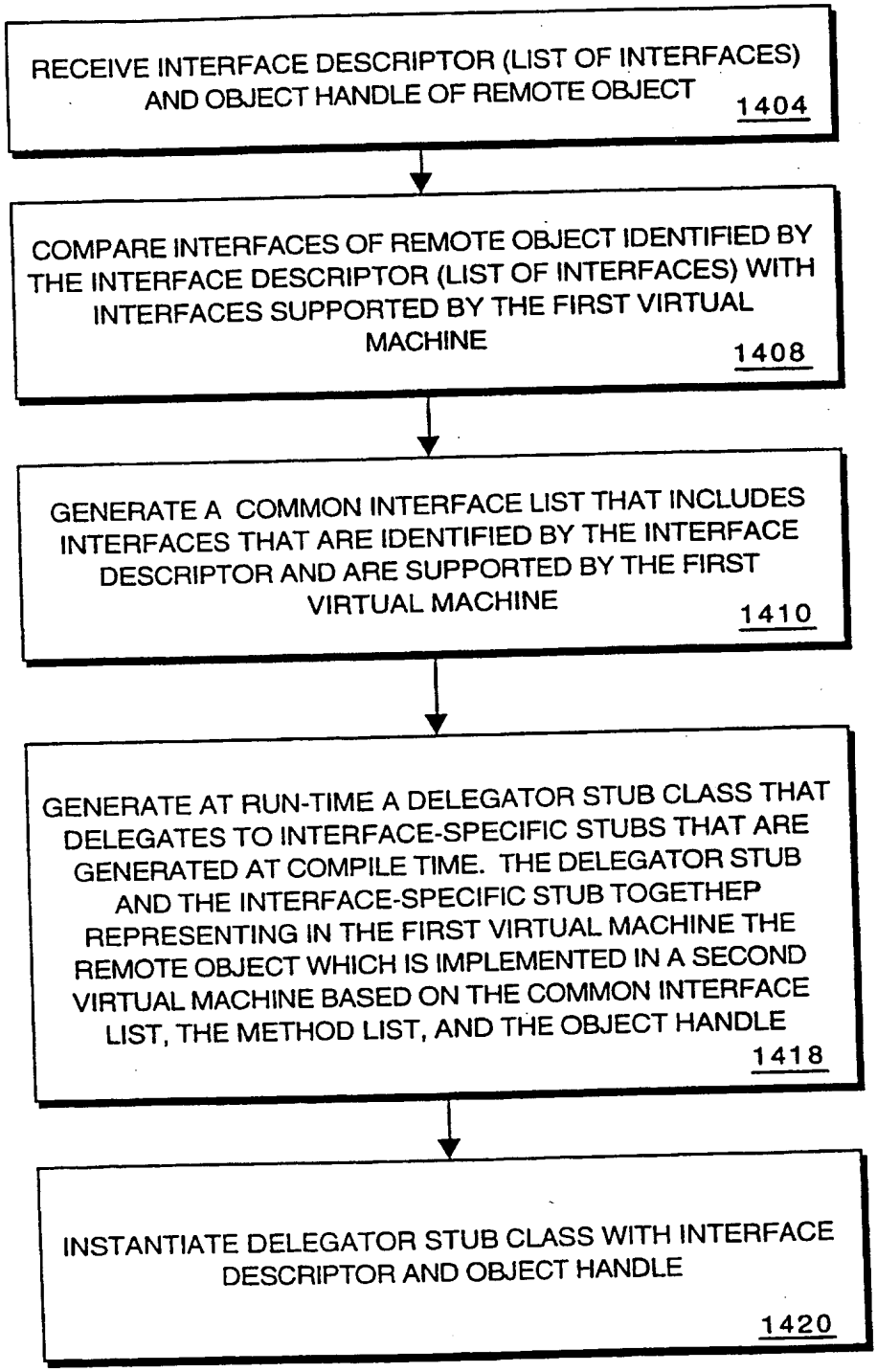


FIG. 15

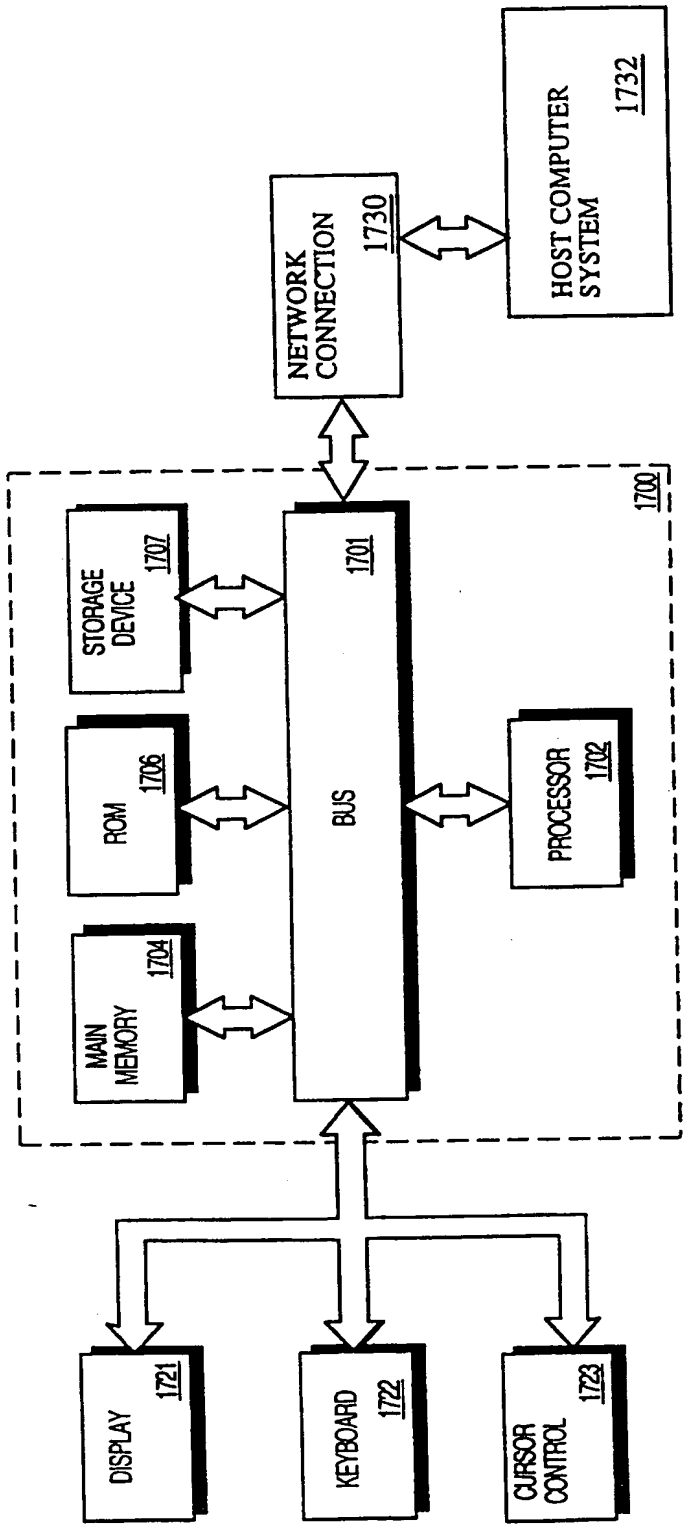


FIG. 17

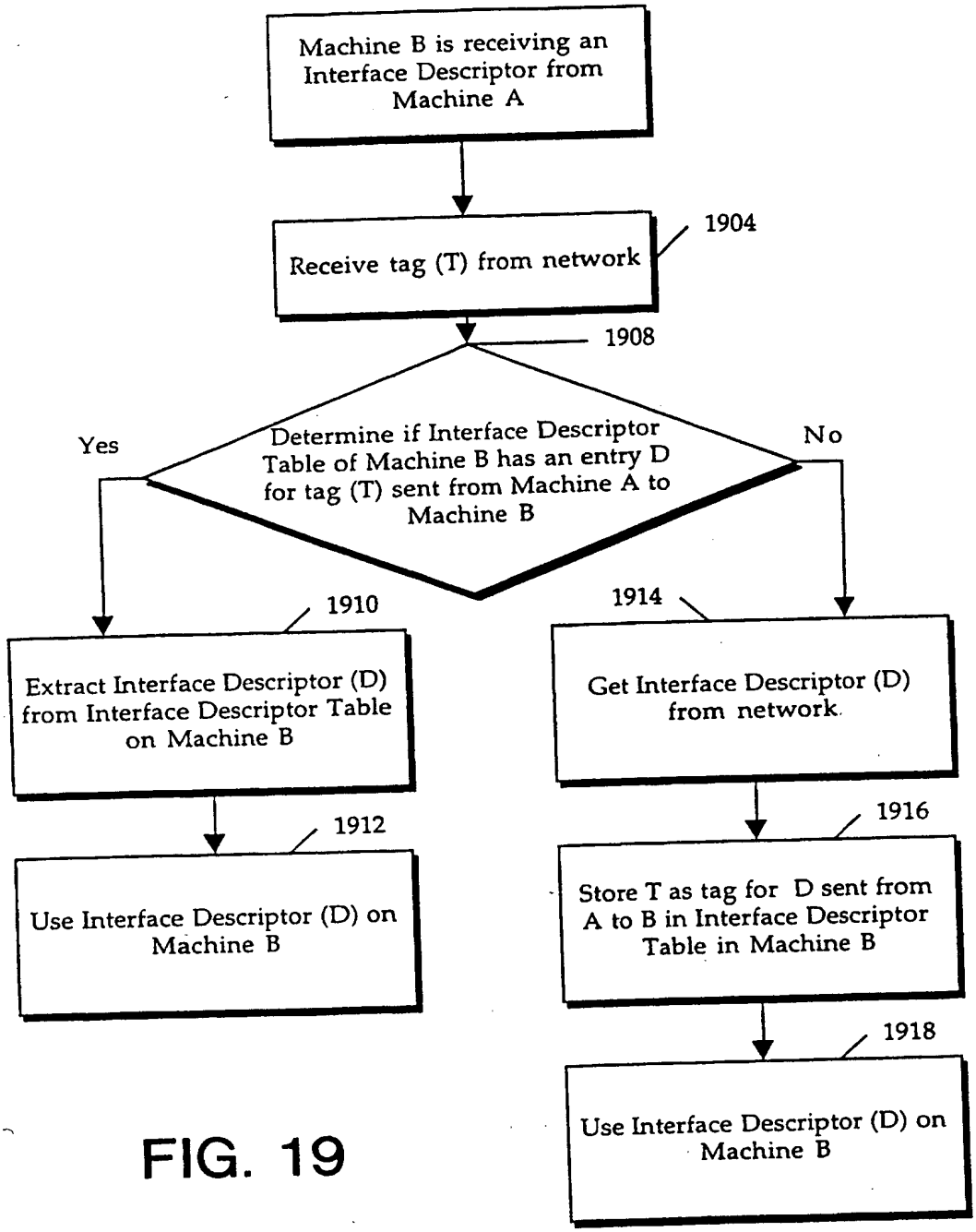


FIG. 19

INTERNATIONAL SEARCH REPORT

information on patent family members

Inte. onal Application No
PCT/US 98/05665

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0643349 A	15-03-1995	JP 7093264 A	07-04-1995
EP 0546684 A	16-06-1993	US 5421016 A	30-05-1995
		JP 5274153 A	22-10-1993